# Convolution Assignment Write-Up

Dahlia Radif

November 28, 2018

## 1 Introduction

In this assignment, we are required to develop a C++ image class that processes a given image and uses convolutions to manipulate the image in a certain way. This class has different methods that blur the input image with a specific kernel, and computes the sharpness of the image with the Laplacian kernel. Below is an in-depth explanation of the functionality of this code.

## 2 Reading in the image

In order to read in the image, the `image.hpp` file defines a Boost array called `image`. Then, we use a constructor in the Image class that takes the image filename as an input and uses the function `ReadGrayScaleJPEG` to read the JPEG file into the Boost array `image`.

## 3 Saving the output image

When saving the output image, we define a method that takes as an input the name of the file to which the image will be saved. If the name of the file is empty, then we set the output file name to be the same as the original file name. Then, the function `WriteGrayScaleJPEG` writes the (possibly modified) to the output file.

## 4 Convolution Method

In this function, we take as inputs three Boost arrays. The first two are populated by unsigned characters, whereby the first is the input image and the second is where we store the convolved image. The third Boost array is populated with floats representing the kernel with which we perform the convolution. We then initialise a variable `outputPixel` of type `double` which is initialised to 0, and will store the convolved value of the pixel.

We then perform two checks. Firstly, if the input and output array have different shapes, then we exit with an error message telling the user that the arrays must be the same size. In addition, we check that the kernel is square and has odd dimensions of at least three. This is because we are required to support these types of kernels in the code. If all of these conditions are satisfied, we then proceed with the convolution.

Firstly, we iterate over each pixel in the input image, and so we have two outer for-loops. For each pixel, we iterate over the elements of the kernel, in order to compute the new value of that pixel after the convolution. To do this, we imagine that we are overlapping the image with the kernel, with the kernel centred over the current pixel. For each overlapping pixel, we multiply its value with the respective kernel element, and the sum of these products gives us the convolved value of the current pixel. Therefore, for each kernel element, the respective pixel image with which it overlaps is indexed by `(i + k - ((kernelRows - 1) / 2)`, `j + m - ((kernelRows - 1) / 2))`, where (i,j) is the index of the current pixel, and (k,m) is the index of the kernel element. However, before we multiply the two values together, we need to ensure that the edges of the input image are processed correctly. For example, for the first pixel of the input image, overlapping a kernel of size 3 centred at this pixel will give out-of-range indices. Therefore, there are four if-statements to handle four different possibilites. Either the row index `i + k - ((kernelRows - 1) / 2)` is less than 0, so we set it equal to 0, or it is greater than the largest row index, so we set it equal to `inputRows - 1` , where `inputRows` is the number of rows in the input array. Similar logic holds if the column index is out of range; if `j + m - ((kernelRows - 1) / 2)` is less than 0, we set it equal to 0, adn if it is greater than the largest column index, then we set it equal to `inputCols - 1` , where `inputCols` is the number of rows in the input array.

After we have ensured that the edge cases are properly handled, we can then compute the convolved pixel value. However, before we add this value to the output array, we take into consideration overflow and underflow. Since the output array contains unsigned characters, the values it holds range from 0 to 255. Therefore, if we obtain values outside of these ranges and do not handle them correctly, we end up with inaccurate results when computing the Sharpness. We therefore implement additional checks; if the output pixel is larger than 255, we set it equal to 255, and if the ouput pixel is less than 0, we set it equal to 0.

## 5 BoxBlur Method

This method takes an input a float representing the size of the kernel, and computes the convolution of the input image with a box blur kernel that results in a blurred imaged. We firstly set the input Boost array to be the image, to ensure that we do not modify the image in place (otherwise the convolution will not be computed correctly). Afterwards, the box blur kernel is created using the input size as the dimensions. Its elements are each 1 / $kernelSize^2$, so that the sum of the elements is 1. We then convolve the image with this kernel. Note that the input Boost array is a copy of the image, whilst the output array is the image object itself, in order to ensure that we write the modified image to the file.

## 6 Sharpness Method

This method returns an unsigned integer that is the maximum of all the pixels in the convolved image. The kernel used is the Laplacian kernel, which is 3x3 and whose structure is specified in the assignment. An output Boost array of the same dimensions as the image is initialised. We convolve the image with the Laplacian matrix, and find the maximum pixel in the output array using `std::max_element`. Note that in this case, the image object is not modified, as we are not required to return the image or use it once we convolve it with the Laplacian operator. However, this can be modified in the future using a similar input Boost array as in the BoxBlur method if such functionality was required. maketitle

## 7 Main Method

In `main`, we create eight instances of the Image class. For the original image, we simply read it, and compute its sharpness and output it. Then, for the other instances, we read the original image, blur it with a specified box blur kernel, save that image to a suitably named file, and then compute and output the sharpness. The output is formatted to match the program specifications.